Approved for Public Release, Distribution Unlimited

FORGES

Formal Synthesis of Generators for Embedded Systems

Program Manager: John Bay

PIs. John Anton, Lindsay Errington

Kestrel Institute

{anton,lindsay}@kestrel.edu

(650) 493-6871

Contract #: F30602-00-C-0155
Award End Date: June 22, 2003

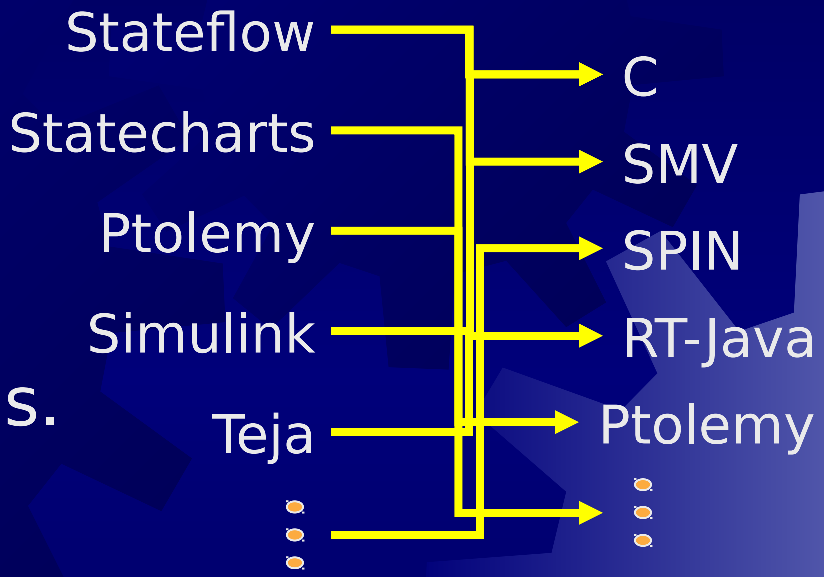
MoBIES PI Meeting
July 24 – 26, 2002
New York

Collaborators

- Berkeley
- SRI
  - Stateflow parser
- Vanderbilt
  - Stateflow semantics
  - HSIF
- Mathworks
- Prospective:
  - CMU - CheckMate

Problem Description

- Require many generators!
- Generators are sophisticated and substantial applications.
- Eliminating errors is difficult.
- Errors are unacceptable!



Semantic mappings!

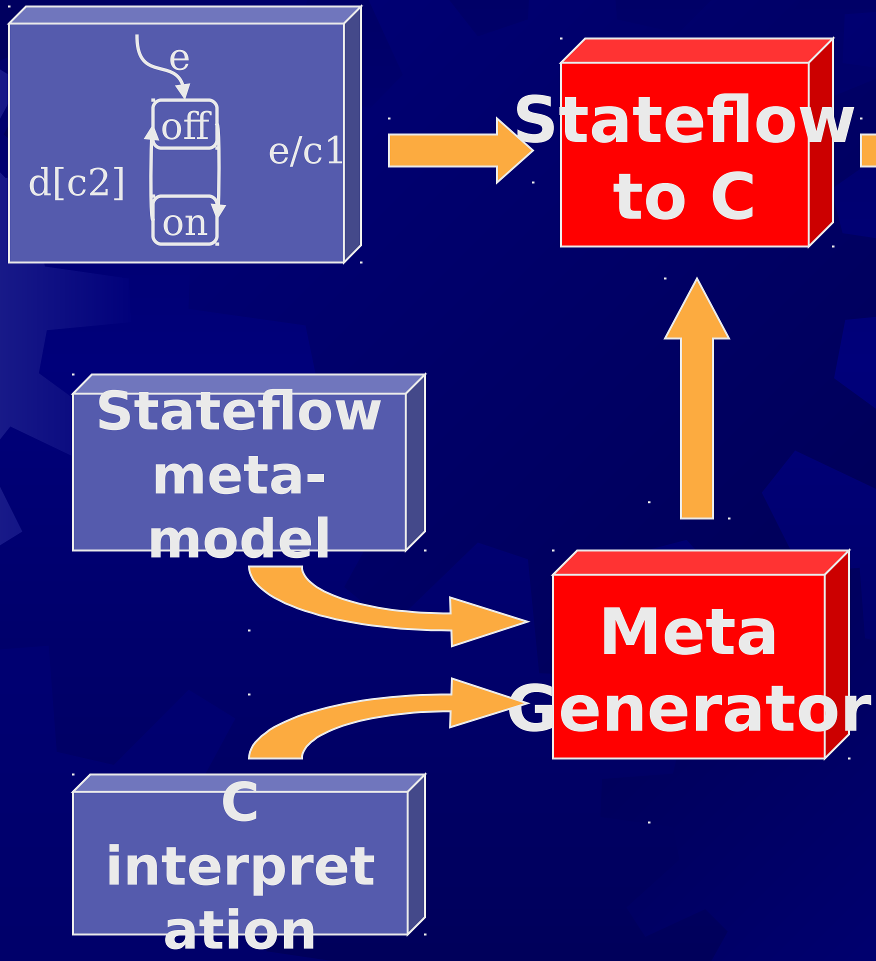
Program Objective

- Synthesize model-based generators!
  - with less effort!
  - that are correct by construction!
  - and yield better code!

Milestone Support

- “Mathematically model generators”
- “Generate embedded software from models”
- “Synthesize generators from formal spec”
- “Guarantee properties of generated systems”

Tool Description



Stateflow to C

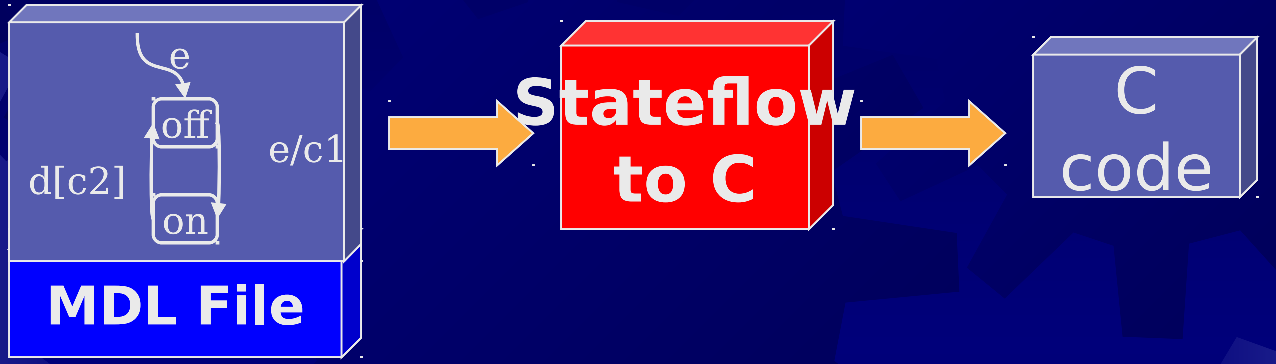
C code

Stateflow meta-model

Meta Generator

C interpretation

Tool Description: Interfaces



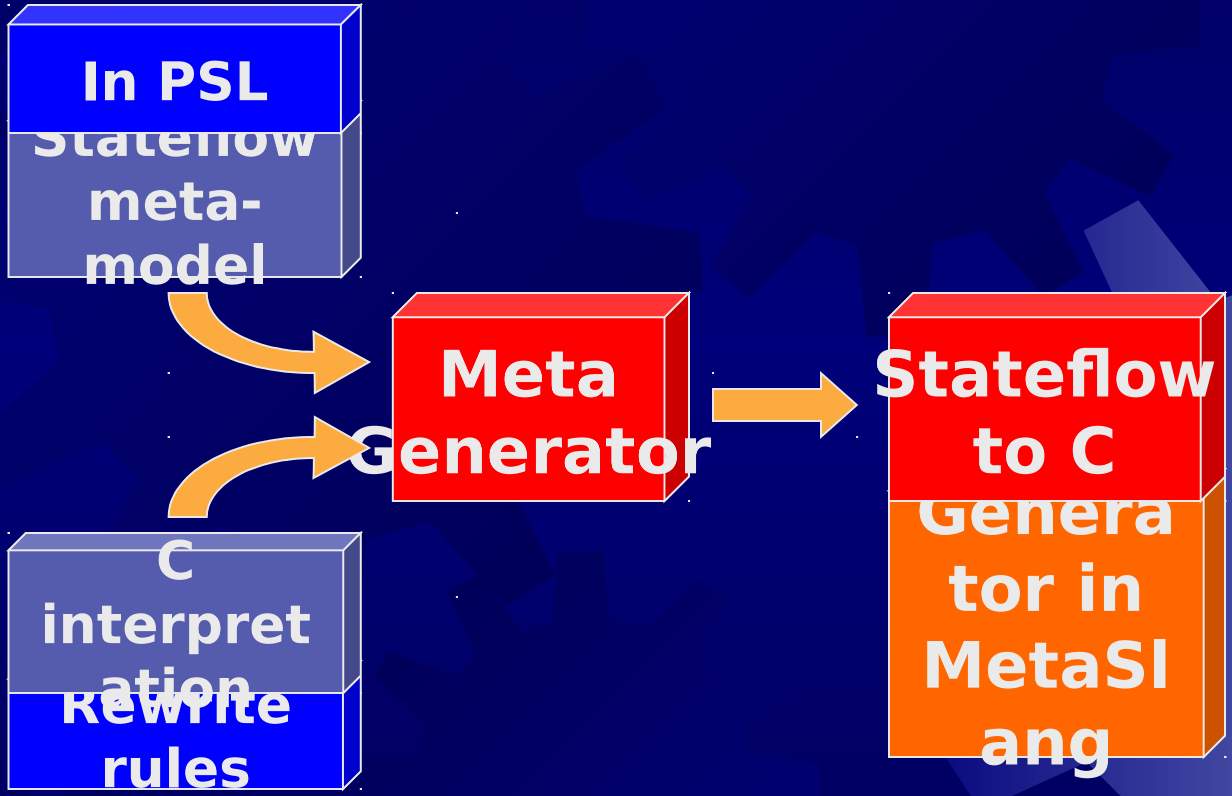
MDL File

Stateflow to C

C code

- For midterm, focus on Stateflow / MT subset
- “Vertical slice” experiments.
- Stateflow/MT omits:
  - state hierarchy
  - junction nodes
  - condition actions

Tool Description: Interfaces



In PSL

Stateflow meta-model

C interpretation

Rewrite rules

Meta Generator

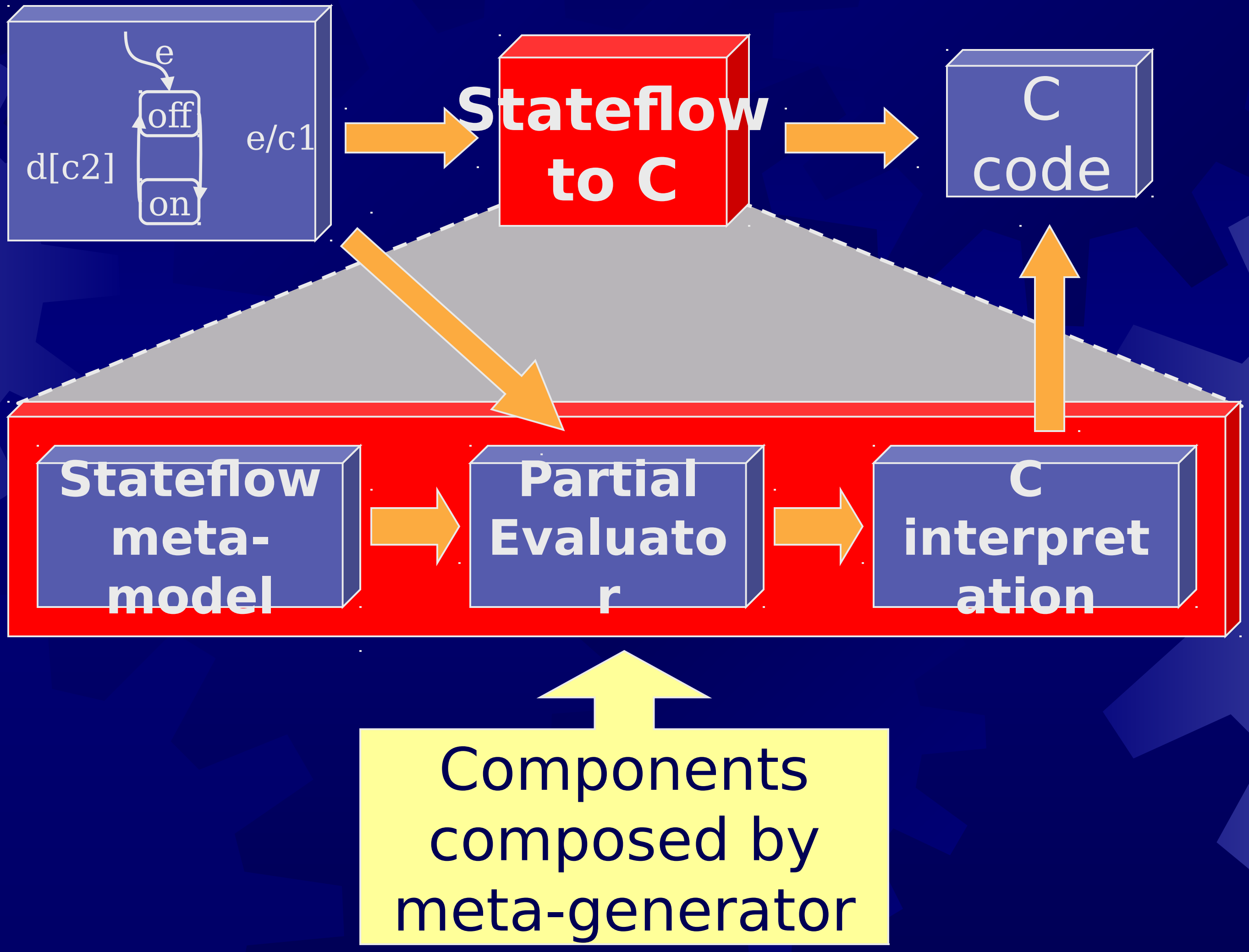
Stateflow to C

Generator in MetaSlang

OEP Participation

- Generators for Berkeley / Ford automotive OEP
- Delivered for midterm:
  - Stateflow / MT meta-model
  - Stateflow / MT → C
- In progress:
  - “Full” Stateflow → C
  - Other generators?: $x \rightarrow y$ *eg* Simulink, HSIF
- OEP contribution:
  - Participation in working group meetings and teleconferences.
  - HSIF
- Technical POC: Jim Misener, Pravin Varaiya, Paul Griffiths, Tunc Simsek

Technical Approach



Stateflow to C

C code

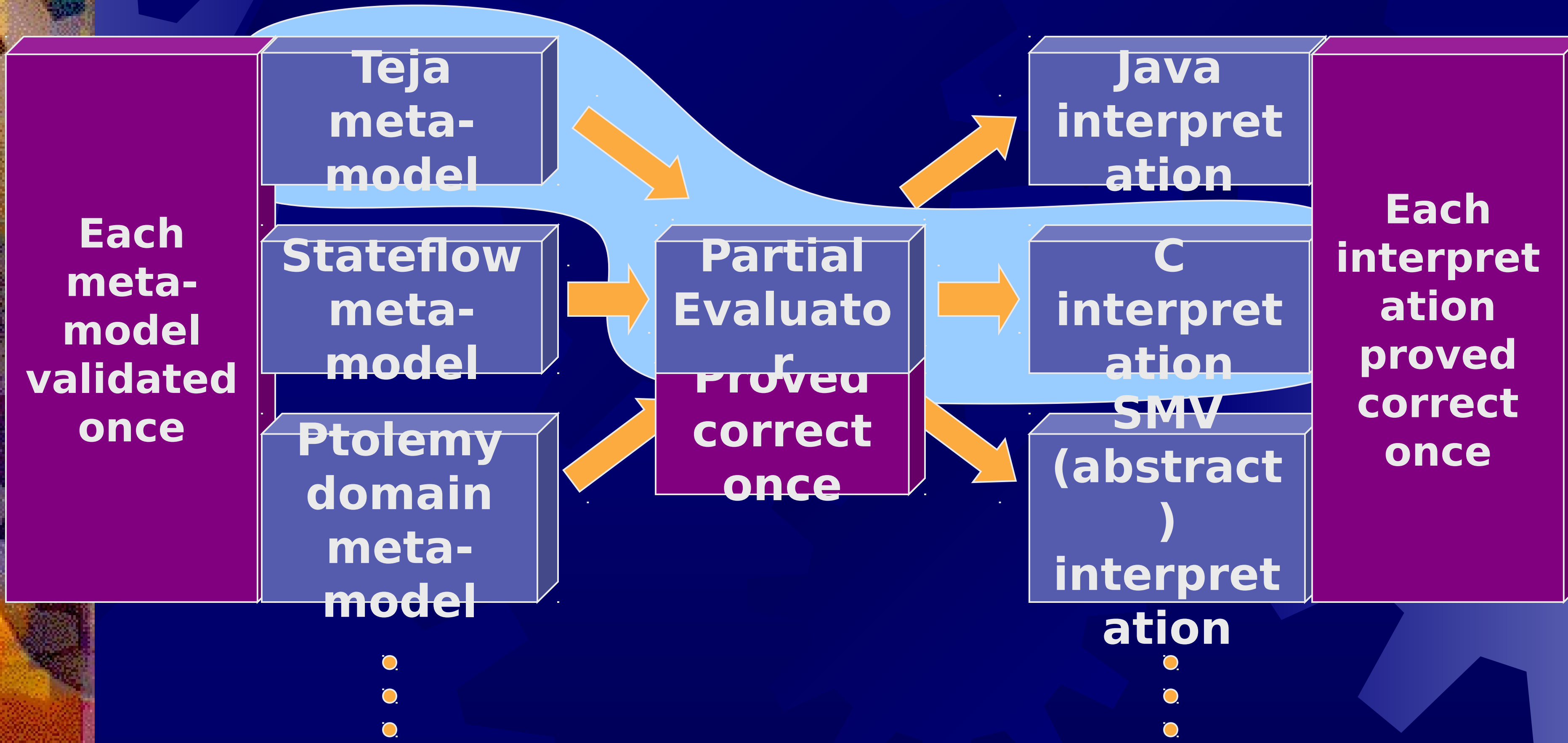
Stateflow meta-model

Partial Evaluator

C interpretation

Components composed by meta-generator

Technical Approach



Each meta-model validated once
Teja meta-model
Stateflow meta-model
Ptolemy domain meta-model
Partial Evaluator
Proved correct once
Java interpretation
C interpretation
SMV (abstract) interpretation
Each interpretation proved correct once

How is the meta-model specified?

- PSL: Procedural Specification Language
- Defines:
  - Static semantics *(what is a well-formed program?)*
  - Dynamic semantics *(how does a program execute?)* = interpreter!

PSL (cont'd)

- Stateflow manual:

Executing an Active State

1. The set of outer flow graphs is executed (see Executing a Set of Flow Graphs). If this causes a state transition, execution stops. (Note that this step is never required for parallel states)
2. During actions and valid on-event actions are performed.
3. The set of inner flow graphs is executed. If this does not cause a state transition, the active children are executed, starting at step 1. Parallel states are executed in the same order that they are entered.

- In PSL

```
executeActiveState(state : State, event : Event)
  transitionTaken := executeFlowGraphs(state.outerTransitions, event)
  if ¬transitionTaken then
      execute(state.duringAction)
      transitionTaken := executeFlowGraphs(state.innerTransitions, event)
```

Partial Evaluation

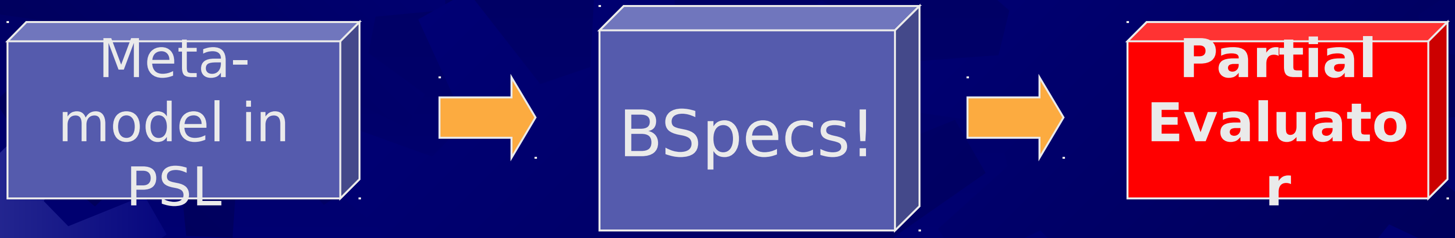
Given algorithm to compute $z = x^y$:

```
let
  var $x$ : $Nat$
  var $y$ : $Nat$
  var $z$ : $Nat$
in
  $z$ := 1;
  while $y \neq 0$ do
        while $2 \mid y$
     do
            $x := x^2$;
             $y := y / 2$;
        $y := y - 1$;
        $z := z \times x$
```

specialize algorithm with $y = 5$

```
let
  var $x$ : $Nat$
  var $y$ : $Nat$
  var $z$ : $Nat$
in
  $z := x(x^2)^2$
```

Problem!



Meta-model in PSL

BSpecs!

Partial Evaluator

- Natural for meta-modeling!
- Straight forward integration with:
  - Stateflow
  - Statecharts
  - Teja
  - Ptolemy domains
  - …

- Must reason about meta-models:
  - specialize, unfold, simplify …
- Transformations must be correct.
- May create unstructured code.
- Needs a simple *semantic representation* for programs!

BSpecs

- “Mathematical” / “Logical” flow-graphs.
- Subsumes other formalisms:
  - Z, Abstract State Machines, transition systems, flow-graphs …
- Hybrid Systems.

PSL → BSpecs

```
let
  var $x : Nat$
  var $y : Nat$
  var $z : Nat$
in
  $z := 1$
  while $y \neq 0$
    do
      while $2 \mid y$ do
        $x := x^2$
        $y := y / 2$
      $y := y - 1$
      $z := z \times x$
```

S = **spec**
 op $x : Nat$
 op $y : Nat$
 op $z : Nat$

$$\begin{tikzcd}
a \arrow[r, "f"] & b \arrow[r, "h"] \arrow[r, bend right, "g"'] \arrow[d, "m"'] & c & k \\
& d & &
\end{tikzcd}$$

$$\begin{tikzcd}
S \arrow[r, "x' = x \wedge y' = y \wedge z' = 1"] & S \arrow[r, "\neg (2 \mid y) \wedge x' = x \wedge y' = y - 1 \wedge z' = z \times x"] \arrow[r, bend right, "y \neq 0 \wedge x' = x \wedge y' = y \wedge z' = z"'] \arrow[d, "y = 0 \wedge x' = x \wedge y' = y \wedge z' = z"'] & S & 2 \mid y \wedge x' = x^2 \wedge y' = y / 2 \wedge z' = z \\
& S & &
\end{tikzcd}$$

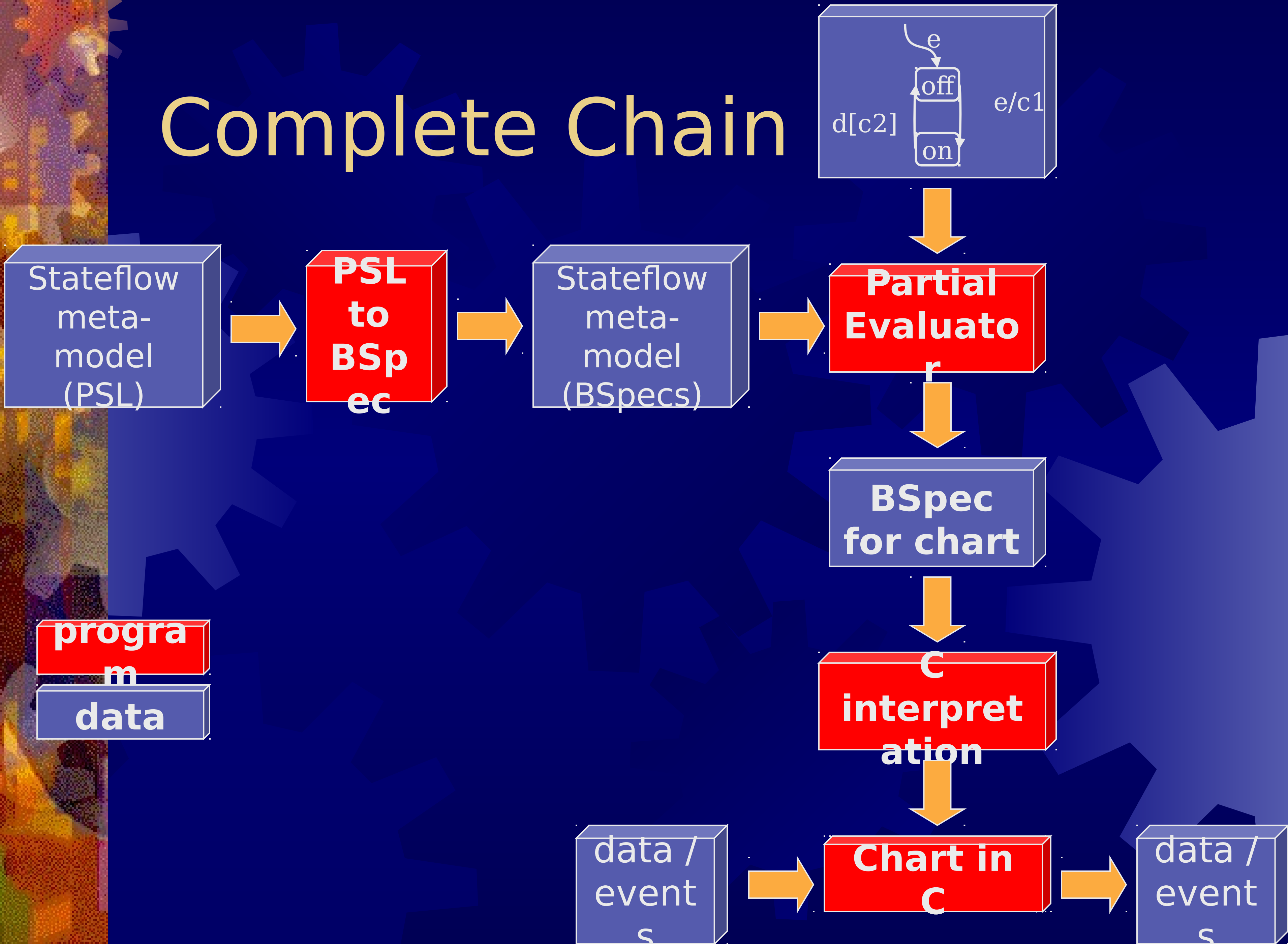
Programs as BSpecs

Computer Science	Category Theory
Flow graphs / State machines / Transition Systems / Kripke frames	BSpecs = Categorical diagrams
Simulation / refinement	Diagram morphisms.
“Models” / unfolding (Including hybrid systems)	“Fibration” / “slice categories”
Constraint propagation / WP semantics	“Adjoint functors”
Partial evaluation / program point specialization	“Fibration”
Parallel composition	“Pushout”

Why does the theory matter?

- Correctness!
- A BSpec is a, simple, mathematically precise representation of “programs”.
- BSpecs facilitate a simple, mathematically precise specification of partial evaluation.

Complete Chain



Stateflow meta-model (PSL)

PSL to BSpec

Stateflow meta-model (BSpecs)

Partial Evaluator

BSpec for chart

C interpretation

data / events

Chart in C

data / events

program

data

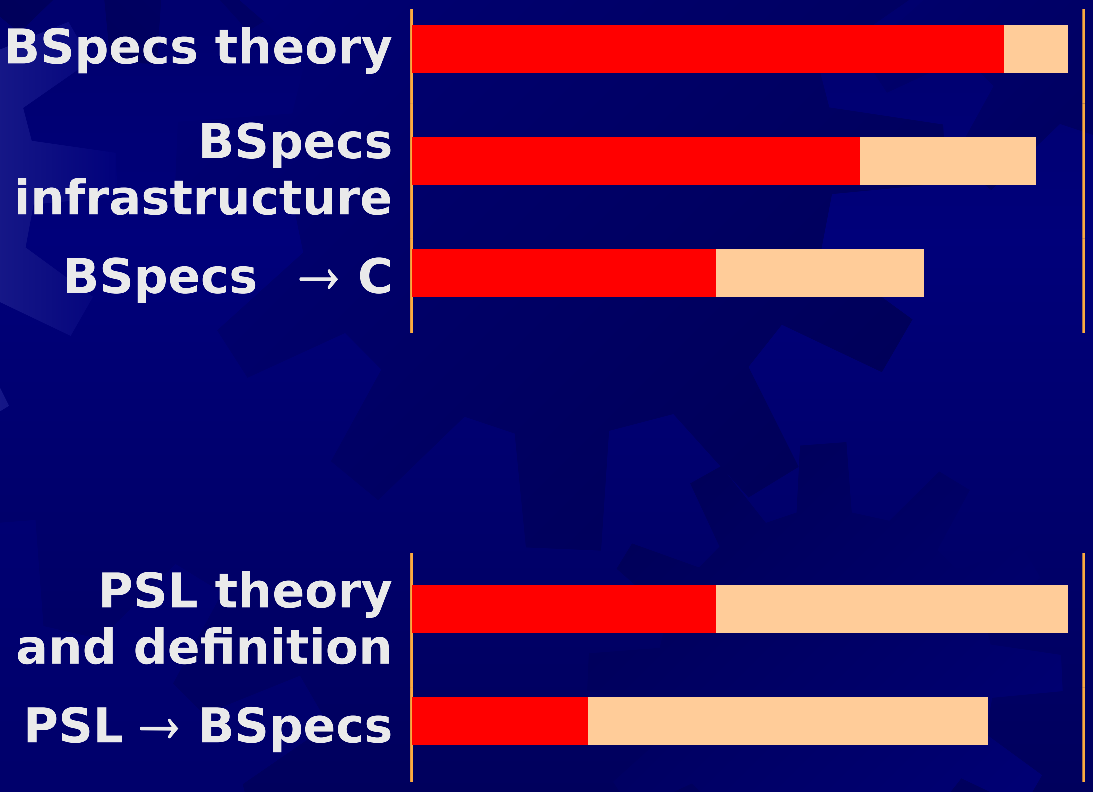
C interpretation

- Partial Evaluation yields a specialized BSpec
- Need: BSpec → C backend
- Simple translation
  - Small “semantic gap”
  - Expressed in rewrite rules.

Project Status

Before last PI meeting 🟥

Since last PI meeting 🟧



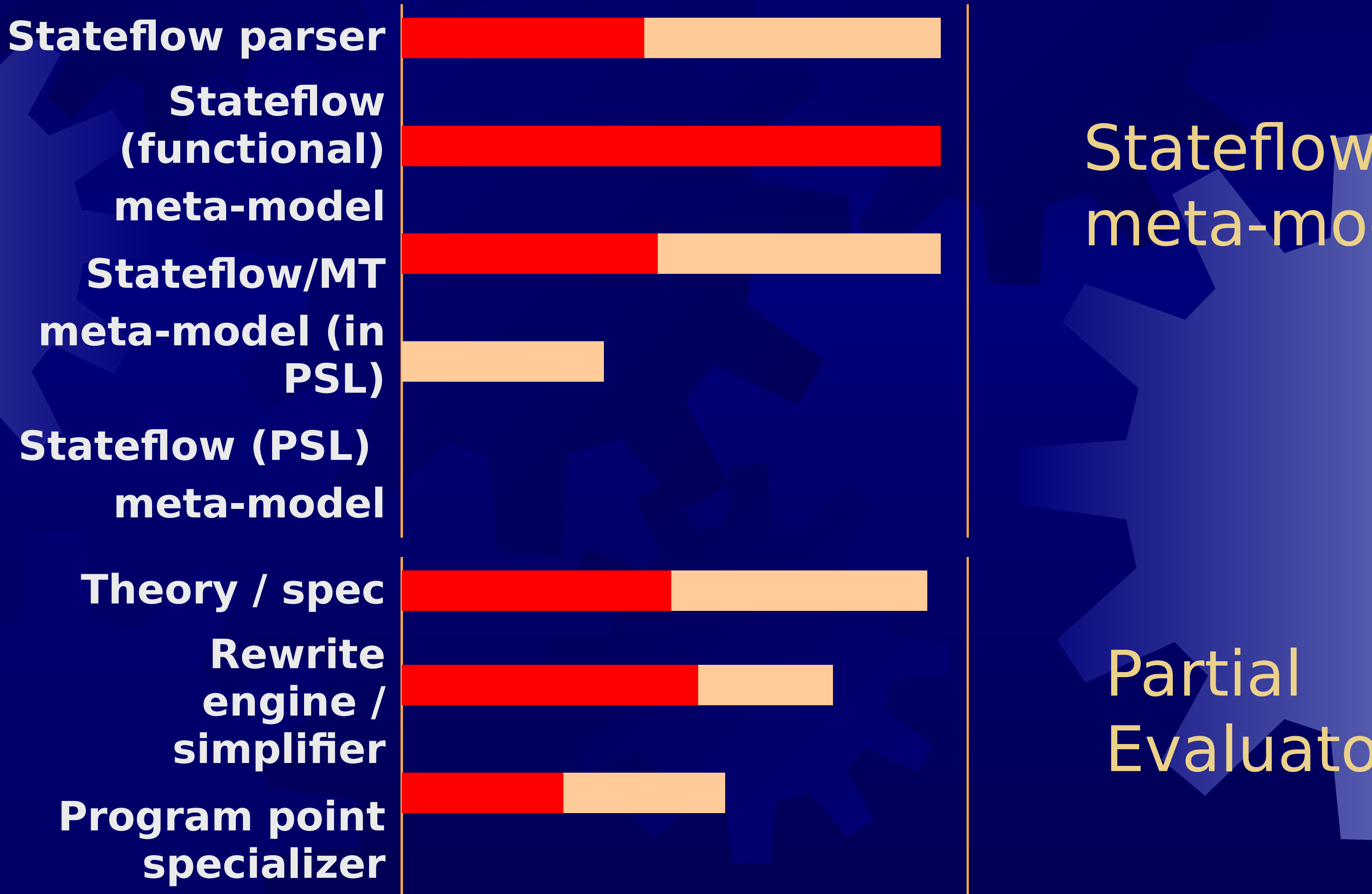
Behavioral
Specificati
(BSpecs)

Procedural
Specificati
Language
(PSL)

Project Status

Before last PI meeting 🟥

Since last PI meeting ▨



Project Plans: Next 6 months

- PSL meta-model for “full” Stateflow
- Consider a second source language:
  - Simulink?
  - HSIF?
- Challenge: performance of partial evaluator

Current PE

- Code is “functional”. No assignment!
- Direct transcription of mathematics into specification and code.
  - Sets are lists
  - Maps are association lists

Improving PE Performance

- PE remains functional
- Refine data-types and algorithms
  - Sets as B-trees, Red-Black trees, etc
  - Maps as trees, hash tables etc.
- 5 – 7 × speedup

Improving PE Performance (2)

- Reimplement partial evaluator in PSL
- Refine data-types and algorithms.
- Update in place: 10 × speedup
- Underlying theory still applies!

Improving PE Performance (3)

- Self application:
  - Specialize the partial evaluator with respect to itself
- “2nd Futamura Projection”
- 7 × speedup

Midterm experiments

- Goal:
  - Stateflow / MT meta-model ✓
  - Stateflow / MT → C generator ✓
- Schedule:
  - Deliver late in March ✓ (April)
- Success criteria:
  - Quality of code ✓
  - Speed of generator ?
  - Adaptability of meta-model ✓

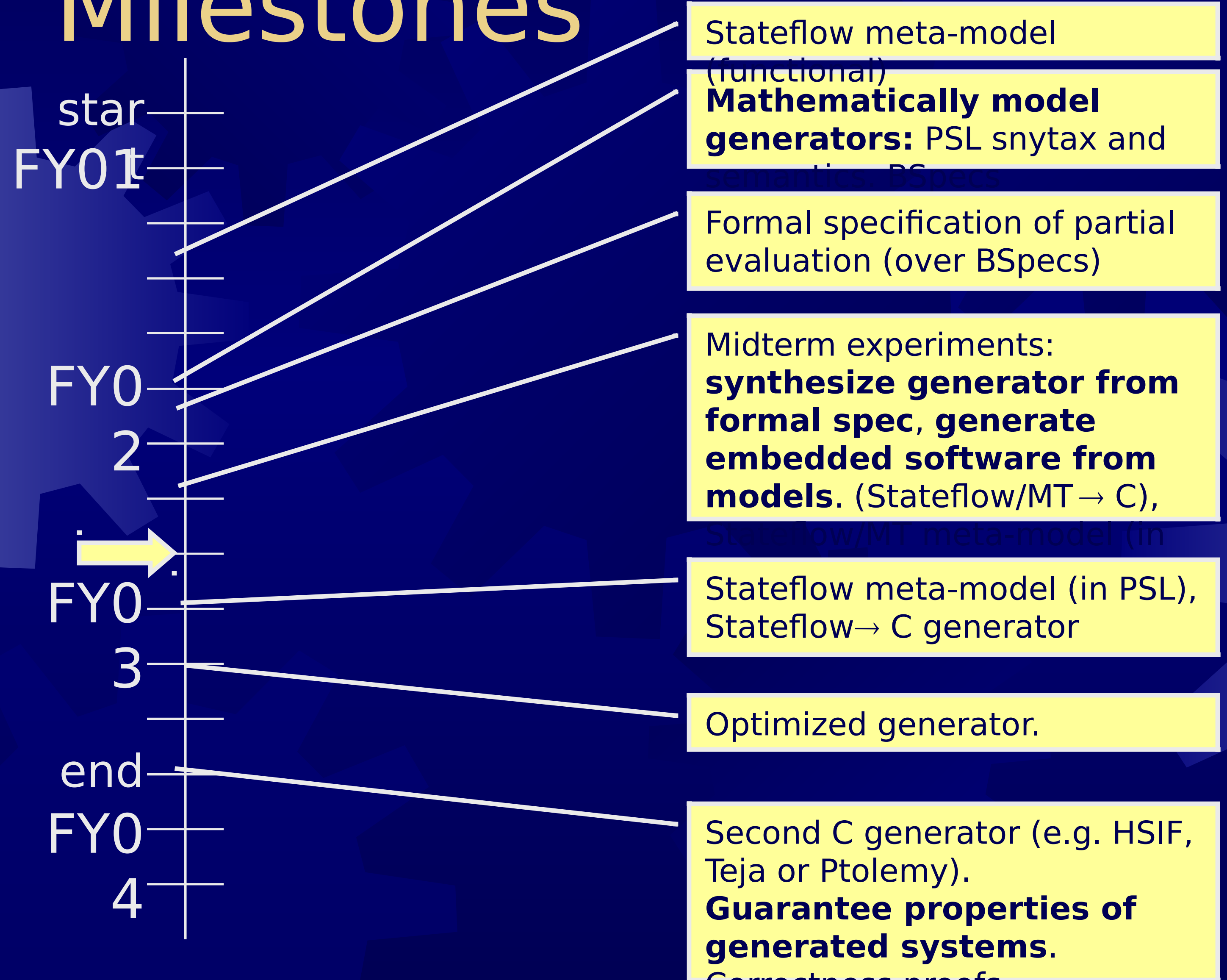
On Adaptability …

- Experiment with Paul Griffiths, Berkeley.
- Misinterpretation of the semantics of "*during actions*".
  - He thought "*during actions*" invoked on every chart activation.
- Paul changed meta-model and successfully produced a second generator.
  - Generated code agreed with his expectation.

Technology Transition/Transfer

- Potential collaboration with Mathworks
- Investigating generators targeting:
  - Java
  - VHDL
- Others:
  - Simulink?
  - HSIF?
  - Teja
  - Ptolemy

Project Schedule and Milestones



Stateflow meta-model (functional)

Mathematically model generators: PSL snytax and semantics. BSpecs

Formal specification of partial evaluation (over BSpecs)

Midterm experiments: **synthesize generator from formal spec**, **generate embedded software from models**. (Stateflow/MT→ C), Stateflow/MT meta-model (in

Stateflow meta-model (in PSL), Stateflow→ C generator

Optimized generator.

Second C generator (e.g. HSIF, Teja or Ptolemy). **Guarantee properties of generated systems**. Correctness proofs

Program Issues

- Evaluation of Meta-Generators?